Microprocessors and Microsystems 39 (2015) 1076–1086

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Hardware/software co-design for a high-performance Java Card interpreter in low-end embedded systems

Massimiliano Zilli [a,*], Wolfgang Raschke [a], Reinhold Weiss [a], Johannes Loinig [b], Christian Steger [a]

[a] Graz University of Technology, Institute of Technical Informatics, Graz, Austria
[b] NXP Semiconductors Austria GmbH, Gratkorn, Austria

ARTICLE INFO

Article history:
Available online 24 May 2015

Keywords:
Hardware/software co-design
Smart card
Java Card
Java interpreter
Hardware-supported interpreter
Application specific instruction set processor

ABSTRACT

Java Card is a Java running environment specific for smart cards. In such low-end embedded systems, the execution time of the applications is an issue of first order. One of the components of the Java Card Virtual Machine (JCVM) playing an important role in the execution speed is the bytecode interpreter. In Java systems the main technique for speeding-up the interpreter execution is the Just-In-Time compilation (JIT), but this resource consuming technique is inapplicable in systems with as restricted resources available as in smart cards.

This paper presents a hardware/software co-design solution for the performance improvement of the interpreter. In the software domain, we adopted a pseudo-threaded code interpreter that allows a better run-time performance with a small amount of additional code. In the hardware domain, we proceeded moving parts of the interpreter into hardware, giving origin to a Java Card interpreter based on an application specific instruction set processor.

1. Introduction

Smart cards are nowadays a widespread technology used mainly in the field of banking, telecommunication, and identification. Typical hardware configurations are based on 8/16 bits processors with some kilobytes of RAM and up to a few hundred kilobytes of non-volatile memory, distributed between ROM, Flash and EEPROM memory. In such small devices, the programming language for the applications development is often C or assembly in order to keep the code size small and the performance high. Issues that arise with these kinds of languages are the portability and the update of the applications. An interpreted language like Java alleviates these problems, also adding a high degree of security to the run-time environment.

Because of the resource constraints in smart cards, the adoption of a complete Java standard is unfeasible. Java Card standard is a reduced version of Java targeted for smart cards [1,2]. Java Card inherits the main features of Java, easing object oriented programming and the *compile once run anywhere* feature. The applications for Java Card are distributed in form of CAP files for the executable binaries and of export files for the interface binaries. The CAP file contains the intermediate code in form of bytecodes that is the result of the Java compilation. In this paper we use the terms "Java bytecode" to indicate the result of the Java compilation, and to distinguish it from the machine instructions (opcodes) that constitute the instruction set of the processor. Once the application has been installed on the smart card, its execution starts and continues until the application is uninstalled.

Like Java, Java Card Virtual Machine has an interpreter whose task is the interpretation of the Java bytecodes. The Java interpreter is a critical part of the Java Card Virtual Machine (JCVM), since it directly affects the execution time of the applications. This aspect is very important in industry, since applications often have very strict requirements. In standard Java, the interpreter has been subject of many optimization techniques aimed to improve the execution speed, but these techniques are not applicable on smart cards, because of the limited resources available.

In this paper, we propose a hardware/software co-design solution that improves the performances of the Java Card interpreter. In the software domain we adopted a pseudo-threaded code solution for the interpreter; in the hardware domain, we integrated parts of the interpreter into the microcontroller architecture. We investigated three solutions with different degrees of integration of the interpreter into the microcontroller architecture and compared them to a classic interpreter implementation.

* Corresponding author.
E-mail addresses: massimiliano.zilli@gmail.com (M. Zilli), wolfgang.raschke@tugraz.at (W. Raschke), rweiss@tugraz.at (R. Weiss), johannes.loinig@nxp.com (J. Loinig), steger@tugraz.at (C. Steger).

http://dx.doi.org/10.1016/j.micpro.2015.05.004
0141-9331/© 2015 Elsevier B.V. All rights reserved.

The structure of the rest of the paper is as follows. Section 2 reviews the previous literature related to this research. Section 4 provides a functional description of the hardware aided interpreter architectures. Section 5 gives an overview of the design and the implementation for the software and hardware part of the Java Card interpreter. Section 6 presents the results on the performance evaluation of the proposed interpreters. In Section 7 we report our conclusions and an outlook on future work.

2. Related work

The interpreter of Java Card Virtual Machine is usually a classic interpreter ideally based on a huge switch wrapped by a while loop [3]. This solution is simple and compact in terms of ROM code size, but suffers from low speed performance. Nevertheless, the classic interpreter is a clean solution that executes hardware independent code, and hence, complies with the *compile once run anywhere* model. An alternative interpreter is the direct threaded interpreter (DTI) [4–6]. This interpreter is based on a compiler that produces a machine depended code. In fact, the executable code consists of a sequence of subroutine addresses that have to be handled; the presence of machine specific addresses into the executable code makes the latter not portable. The portability problem is solved in [5] separating the compilation of the application in two phases: the first produces a preliminary code that is portable, while the second creates the threaded code.

An application of the DTI to the Java System is proposed in [7]. Analogously to [5], the problem of the portability is overcome by adopting a pre-execution phase that transforms the Java bytecode into threaded code. This solution hardly fits into a Java Card environment, because of the remarkable increase of the executable code size (i.e. in an architecture with 16-bit wide address space, the executable threaded code would be about twice the size of the original bytecode).

The main approach to reduce execution time present in most widely spread Java environments is the *Just-In-Time* (JIT) compilation [8–11]. Instead of being interpreted, the bytecodes are compiled into machine code performing optimizations that make their execution faster than a normal interpretation. Although this mechanism is very effective in general purpose systems and high-end embedded systems, it is not applicable in smart cards because of the high amount of RAM it needs for storing the compiled code.

To overcome the issue of limited resources in smart cards, some authors have proposed alternative methods. Azevedo et al. introduced an *Annotation Aware Virtual Machine* able to recognize annotations that indicate foldable sequences [12]. Hence, during the execution, when the virtual machine encounters an annotation, it executes the superoperator relative to the foldable sequence instead of the normal bytecode sequence. Since the superoperator is an optimized form of the bytecode sequence, the execution is faster than in the plain interpretation.

Another direction in which research has gone for enhancing performance in Java is the hardware implementation of the Java Virtual Machine. The hardware acceleration can be achieved in two main ways, a direct Java bytecode execution or a Java bytecode translation. McGham et al. proposed picoJava [13], a Java processor that executes Java bytecodes directly on hardware. The Java virtual machine is completely implemented in hardware and the Java bytecodes constitute the instruction set of the processor. In this model there is no longer an interpreter, because the processor executes the Java bytecode natively, with outstanding runtime performance. A problem of this architecture is the integration with established operating systems or existing applications, since the processor is not able to execute programs written in other programming languages. An example of bytecode translation into native machine op-code sequences is ARM Jazelle [14]. The integration of the Jazelle with existing operating systems and the concurrent execution of other applications written in other programming languages is possible using the extended instruction set.

The instruction set extension is common practice in application specific instruction set processor (ASIP) [15,16]. With the new opcodes of the instruction set, it is indeed possible to activate hardware functionalities added for the purpose of enhancing the performance of a specific application. In the solution that we propose in this paper we extended the instruction set of a microcontroller to support a pseudo-threaded interpreter whose fetch–decode part is executed in hardware. The execution phase of the Java bytecodes is kept in software for taking advantage of its flexibility. The latter is indeed necessary, for example, to add security checks inside the Java bytecode functions [17].

3. The architecture

This section presents a view over the architecture of the Java Card technology to permit the reader to further easily identify the parts object of this work. Smart cards can be seen as the composition of a hardware part (the microcontroller) and a software part. Usually, the latter is composed by two parts, the firmware layer and the applications. In embedded systems, to collect the functionalities commonly used by the applications and to run more than one application at once, the firmware layer often reaches such a level of complexity that it can be considered an operating system. In the smart card context, the use of such small operating systems mitigates the problem of the portability from one platform to another but does not offer a standard development for third parties yet. The use of a software architecture based on an interpreter like Java Card offers an hardware independent platform that makes the application development easier for third parties.

For smart cards enabled with Java Card, the Java Card run-time environment behaves like an operating system running the applications. The entire system architecture of the smart card can be sketched into a four layers structure as shown in Fig. 1. The upper layer is the application layer and consists of all the Java applications running on the Java Card Virtual Machine. The Java Card APIs represent a standard interface between the applications and the Java Card Virtual Machine. In this way, the Java Card Virtual Machine layer can be considered as an operating system with its APIs. The most relevant components inside the virtual machine are the loader, the security manager and the bytecode interpreter. The role of the latter consists of translating the bytecodes constituting the Java applications into machine instructions. Below the Java Card Virtual Machine layer there is the operating system of the smart card, which offers all the functionalities needed by the virtual machine. The hardware layer consists of the

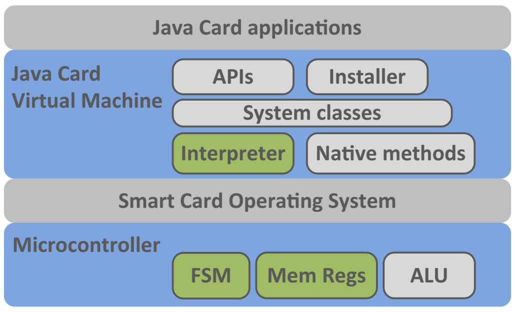

Fig. 1. Architecture of a smart card enabled with Java Card.